Lesson 08

集合
Collections

List

结构

名	解释
List	接口
LinkedList	类，链表实现
ArrayList	类，基础数组实现

数组 转 List

```
String[] arr = {"6", "7", "3", "8", "4"};
List<String> list = Arrays.asList(arr);
System.out.println(list);
```

元素为基础类型时，需要额外一步：

需要 Apache Common Lang 将其转为对象数组

```
int[] arr = {6, 7, 3, 8, 4};
Integer[] objects = ArrayUtils.toObject(arr);
List<Integer> list = Arrays.asList(objects);
System.out.println(list);
```

排序
Sorting

排序

```java
List<Integer> list = new LinkedList<>();
list.add(6);
list.add(7);
list.add(3);
list.add(8);
list.add(4);
Collections.sort(list);
System.out.println(list);
```

有可能会排序成功的，基础类型 + String

使用 Comparator

写一个类 实现 Comparator 接口：

SpecialComparator

```java
public class SpecialComparator implements Comparator<Integer> {
    @Override
    public int compare(Integer o1, Integer o2) {
        int d1 = o1 % 10;
        int d2 = o2 % 10;
        if (d1 < d2) {
            return -1;
        } else if (d1 > d2) {
            return 1;
        } else {
            return 0;
        }
    }
}
```

使用 自定义 Comparator：

```java
List<Integer> list = new LinkedList<>();
list.add(16);
list.add(7);
list.add(3);
list.add(8);
list.add(4);
Collections.sort(list, new SpecialComparator());
System.out.println(list);
```

Comparator 内部机制

Comparator 默认效果是 让排序后升序

通过比较两个值，你的返回值在传递以下信息

返回值	意义
-1	现在是升序，无需调整
1	现在是降序，需要位置互换
0	两个相等

挑战：

Point 根据 x 进行排序， x 相同，则对 y 排序

更好的 Comparator 套路

1. 写一个函数，让每一个要比较的值，都变为一个整数

2. 调用对应类型的 compare 方法

```
public class SpecialComparator implements Comparator<Integer> {
    @Override
    public int compare(Integer o1, Integer o2) {
        return Integer.compare(value(o1), value(o2));
    }

    private int value(Integer o) {
        return o % 10;
    }
}
```

挑战：

Point 根据 x 进行排序

逆序

```
List<Integer> list = new LinkedList<>();
list.add(16);
list.add(7);
list.add(3);
list.add(8);
list.add(4);
Comparator<Integer> c = new SpecialComparator();
c = c.reversed();
Collections.sort(list, c);
System.out.println(list);
```

组合

写两个比较器，然后组合在一起

SpecialComparator2

```
public class SpecialComparator2 implements Comparator<Integer> {
    @Override
    public int compare(Integer o1, Integer o2) {
        return Integer.compare(value(o1), value(o2));
    }

    private int value(Integer o) {
        return o / 10;
    }
}
```

```
List<Integer> list = new LinkedList<>();
list.add(36);
list.add(16);
list.add(3);
list.add(48);
list.add(28);
Comparator<Integer> c = new SpecialComparator().thenComparing(new SpecialComparator2());
Collections.sort(list, c);
System.out.println(list);
```

挑战：

Point 根据 x 进行排序

使用 Comparable

如果 被排序的类型 实现了 Comparable 接口，则可以在排序时 不提供 Comparator

```java
public class Point implements Comparable<Point> {
    int x;
    int y;

    ...

    @Override
    public int compareTo(Point o) {
        int result = Integer.compare(x, o.x);
        if (result == 0) {
            result = Integer.compare(y, o.y);
        }
        return result;
    }
}
```

```java
List<Point> list = new LinkedList<>();
list.add(new Point(3, 4));
list.add(new Point(3, 5));
list.add(new Point(2, 6));
list.add(new Point(2, 5));
Collections.sort(list);
System.out.println(list);
```

集合
Set

是什么

签名板

有元素，不重复

无索引，不保留顺序

基本

```java
Set<String> terms = new HashSet<>();
terms.add("A");
System.out.println(terms.contains("A"));
// terms.get(?)
```

无法 get，因为没有索引

添加重复

```
Set<String> terms = new HashSet<>();
terms.add("A");
terms.add("A");
System.out.println(terms.size());
```

删除

```
Set<String> terms = new HashSet<>();
terms.add("A");
terms.add("A");
terms.remove("A");
System.out.println(terms.contains("A"));
```

遍历

只能使用 for in

```
Set<String> terms = new HashSet<>();
terms.add("Z");
terms.add("Z");
terms.add("A");
terms.add("X");
for (String term : terms) {
    System.out.println(term);
}
```

映射

Map

什么是

存包处

键值对，存储时附带key

key 不能重复，value 可以重复

基本

```java
Map<String, Integer> map = new HashMap<>();
map.put("A", 1);
map.put("B", 1);
System.out.println(map.get("A"));
```

key 一般用 String 或者 Integer

重复添加

只会保留最后一次的信息

```java
Map<String, Integer> map = new HashMap<>();
map.put("A", 1);
map.put("A", 2);
System.out.println(map.size());
System.out.println(map.get("A"));
```

删除

```java
Map<String, Integer> map = new HashMap<>();
map.put("A", 1);
map.put("A", 2);
map.remove("A");
System.out.println(map.size());
System.out.println(map.get("A"));
```

Key 遍历

获取 所有的 Key，构成 Set，然后遍历

```java
Map<String, Integer> map = new HashMap<>();
map.put("A", 1);
map.put("B", 2);
Set<String> keys = map.keySet();
for (String key : keys) {
    System.out.println(key + " -> " + map.get(key));
}
```

Entry 遍历

获取 所有的 键值对 Entry，构成 Set，然后遍历

```java
Map<String, Integer> map = new HashMap<>();
map.put("A", 1);
map.put("B", 2);
Set<Map.Entry<String, Integer>> entries = map.entrySet();
for (Map.Entry<String, Integer> entry : entries) {
    System.out.println(entry.getKey() + " -> " + entry.getValue());
}
```

Value 遍历

获取 所有的 Value，构成 Collection，然后遍历

```java
Map<String, Integer> map = new HashMap<>();
map.put("A", 1);
map.put("B", 2);
Collection<Integer> values = map.values();
for (Integer value : values) {
    System.out.println(value);
}
```

时间

Date and Time

基础概念

时间点

Timestamp

我们同一时刻，都在同一个时间点

计算机用 milliseconds since January 1, 1970, 00:00:00 GMT

相对时间（时区化时间）

几点几分几秒

同一个时间点下，不同时区，是不同的时间

日期

哪年哪月哪日

日期基于时区，纽约的早上，北京的晚上

日期基于日历，阳历，阴历

旧版时间系统

获取当前时间戳

```
System.out.println(System.currentTimeMillis());
```

获取当前时间

```
Date date = new Date();
System.out.println(date);
System.out.println(date.getTime());
```

获取当前时区小时

Calendar 小语法

```
Calendar calendar = new GregorianCalendar();
calendar.setTime(new Date());
System.out.println(calendar.get(Calendar.HOUR_OF_DAY));
```

切换时区

```
Calendar calendar = new GregorianCalendar();
calendar.setTime(new Date());

calendar.setTimeZone(TimeZone.getTimeZone("GMT+08"));
System.out.println(calendar.get(Calendar.HOUR_OF_DAY));

calendar.setTimeZone(TimeZone.getTimeZone("GMT-04"));
System.out.println(calendar.get(Calendar.HOUR_OF_DAY));
```

整体格式化

Format 小语法

```
SimpleDateFormat format = new SimpleDateFormat("HH:mm:ss");
System.out.println(format.format(new Date()));
```

新版时间系统

获取当前时间点

```
Instant instant = Instant.now();
System.out.println(instant);
```

获取当前时间

```
ZonedDateTime now = ZonedDateTime.now();
System.out.println(now);
```

切换时区

```
ZonedDateTime now = ZonedDateTime.now();
now = now.withZoneSameInstant(ZoneId.of("Asia/Shanghai"));
System.out.println(now);
```

获取时间部分

```
ZonedDateTime now = ZonedDateTime.now();
System.out.println(now.getHour());
```

时间计算

```
ZonedDateTime now = ZonedDateTime.now();
now = now.withDayOfMonth(8);
now = now.plusHours(10);
System.out.println(now);
```

去时区概念

```
LocalDateTime dateTime = LocalDateTime.now();
LocalDate date = LocalDate.now();
LocalTime time = LocalTime.now();
```

计算差值

```
ZonedDateTime now = ZonedDateTime.now();
ZonedDateTime before = now.withDayOfMonth(8).plusHours(10);
Duration duration = Duration.between(before, now);
System.out.println(duration);
Period period = Period.between(before.toLocalDate(), ZonedDateTime.now().toLocalDate());
System.out.println(period);
```